A Summary of NN, RNN, XGBoost

Hengtao Zhang

June 29, 2018

1 An Introduction to Neural Network

To better understand Neural Network (NN), we start from the simplest network with only one neuron. Such a naive case could naturally be derived from common linear regression model. Given n independent sample pairs $(\boldsymbol{x}_i, y_i)_{i=1}^n \in \boldsymbol{R}^{p+1}$, where $\boldsymbol{x}_i \in \boldsymbol{R}^p$ and $y_i \in \boldsymbol{R}$, we assume data satisfying following linear expression[1]:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \quad i = 1, \ldots, n.$$

And the above equations can be represented by the graph as shown in Figure 1. Compared with linear model, the single neuron adds an extra activation layer when it comes to the output value. To be more specific, for every sample, we have

$$\begin{aligned} z(\boldsymbol{x}) &= \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \\ y &= a(z) = \sigma(z(\boldsymbol{x})), \end{aligned}$$

where $\sigma(\cdot)$ indicates activation function. It's trivial that the neural network will be reduced to typical linear regression when $\sigma(z) = z$. In practice, people usually choose sigmoid or tanh or ReLU function to activate the linear combinations $z(\boldsymbol{x})$.

In more general case, the NN may contain several hidden layers and multiple neurons may also be stacked in each layer, but the mechanism and intuition are all the same. Figure 3 is a representative example, and it's easy to derive the model formula in a tensor manner if we follow the idea shown in aforementioned cases.

Apart from derivations, it remains an open issue that how the neural network works. Christopher Olah provides a perspective[2] based on manifold and topology to interpret neural network. I also attach my reproduced code in a baby classification case on github for you to see.

[1] You may find more background knowledge from Andrew Ng's online courses

[2] `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`

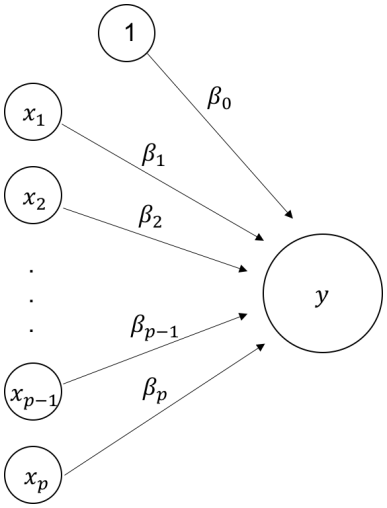


Figure 1: Linear Regression Model

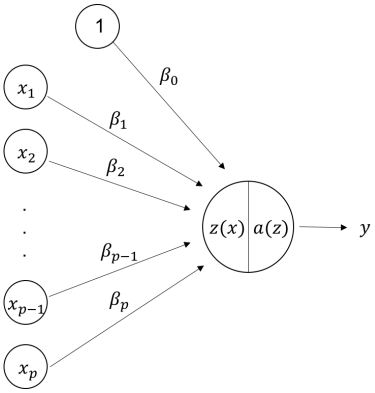


Figure 2: NN with Single Neuron

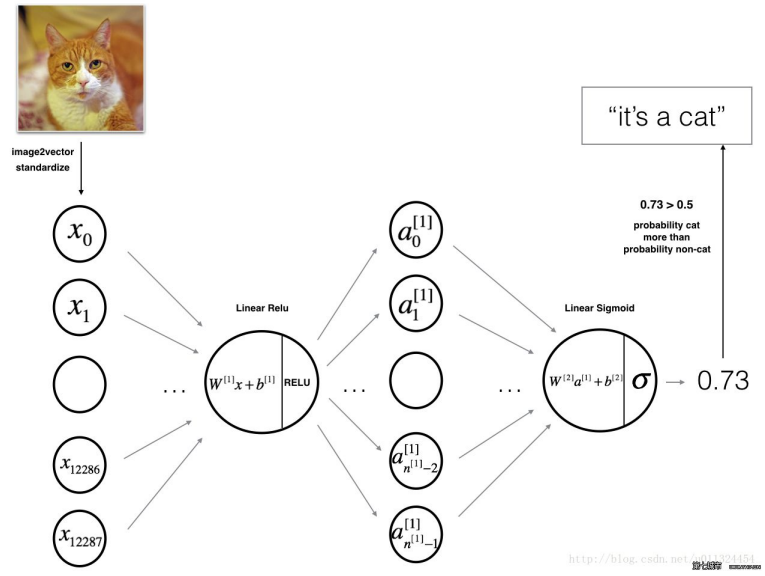


Figure 3: A General Neural Network

2 RNN, LSTM and GRU

Still, we begin with a single neuron of Recurrent Neural Network (RNN). But before formal introduction, I want to highlight the data representation differences between NN and RNN firstly[3].

[3]The contents regarding data representation mostly come from the book *Deep Learning with Python*.

As for NN, the data are usually encoded as a 2D tensor or a matrix shown as the following table:

Response	Feature 1	Feature 2	...	Feature $p-1$	Feature p
y_1	x_{11}	x_{12}	...	x_{1p-1}	x_{1p}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
y_n	x_{n1}	x_{n2}	...	x_{np-1}	x_{np}

Table 1: A 2D Data Tensor

When time or other sequence order matters in data, each sample will be encoded as a 2D tensor and the data therefore be represented by a 3D tensor (see Figure 4). The response variable here could be either vector or 2D tensor correspondingly.

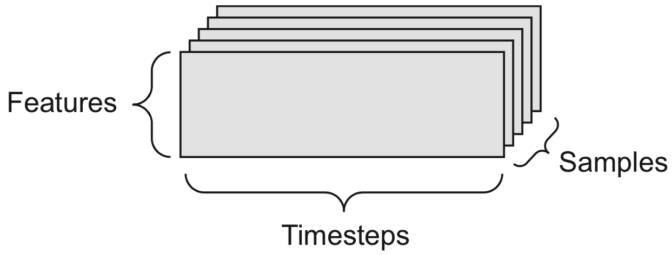


Figure 4: A 3D Timeseries Data Tensor

Simple RNN Reminding that NN proceeds a vector for each sample, so the RNN similarly should own a proper structure to digest the 2D tensor. Denote the 2D tensor as $\boldsymbol{X}^{(i)} = (\boldsymbol{x}_1^{(i)}, \boldsymbol{x}_2^{(i)}, \ldots, \boldsymbol{x}_{T_x}^{(i)})$, where i indicates the sample number while $\boldsymbol{x}_t \in \boldsymbol{R}^p$. To simplify the notation, we neglect superscript (i) in the rest of this report. With above preliminaries, the RNN neuron can be unrolled as displayed in Figure 5

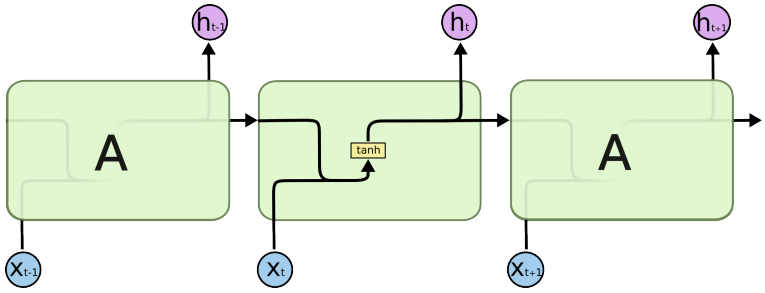


Figure 5: Unrolled RNN Structure

We hereby obtain the recursive formula for hidden states of RNN,

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h).$$

The weights W_{hx} and W_{hh} as well as bias b_h are shared in all time steps, so RNN takes time periods into account. However, the effective coverage of RNN is short-term, that is, only recent steps really matter for current state deciding. To resolve this, two new variants of RNN were introduced, which are LSTM and GRU respectively[4].

LSTM Generally, Long Short Term Memory (LSTM) adds another cell state to remember long-term memory and uses gate function to adjust it intelligently (See Figure 6).

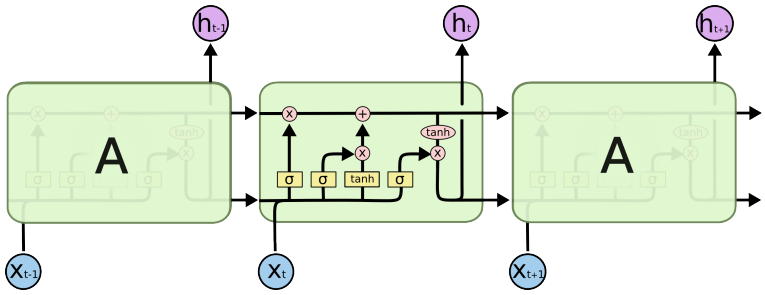


Figure 6: The Repeating Module of LSTM

For the sake of convenience, we give following notations,

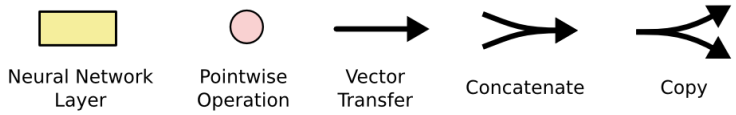
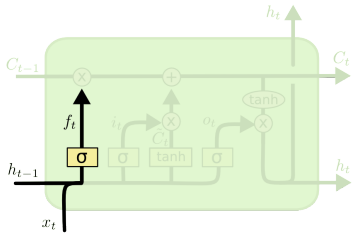


Figure 7: The Notation in LSTM Descriptions

[4]You may refer to Colah's blog for detailed introductions.

Moreover, we denote $[h_{t-1}, x_t]$ as vector concatenation in row, that is, $[h_{t-1}, x_t] = \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$. Now we can introduce the main idea of LSTM step by step.

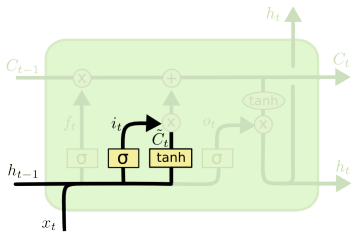
Figure 8 shows the forget gate in LSTM, which is calculated by a single layer NN with the sigmoid activation. Such gate control the percentage of remained information in previous cell state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

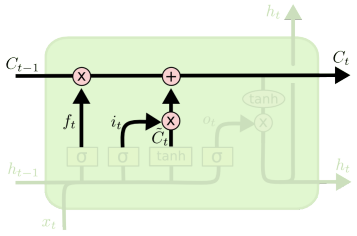
Figure 8: 'Forget Gate' in LSTM

Intuitively, we need to extract new cell message from x_t and h_{t-1}, and blend it with old one through an update gate. (See Figures 9 & 10)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

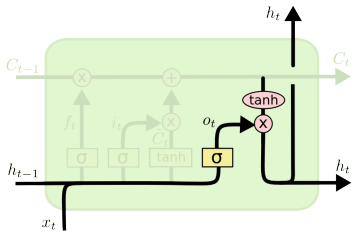
Figure 9: 'Update Gate' in LSTM



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 10: Cell State Update in LSTM

Finally, the hidden state will be updated via new C_t and another output gate will be waked to regulate terminal result. (See Figure 11)

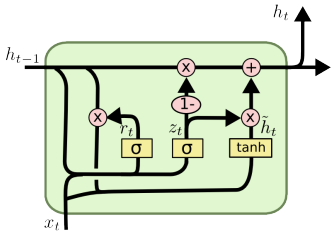


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 11: 'Output Gate' in LSTM

To summarize, there are totally three gates in LSTM and all of them are determined by h_{t-1} and x_t. As for the placement, three gates are all set after related activations. Additionally, long-term memory are stored in cell state and adopted to update short-term memory which locates in h_t. Absorbing the gate idea mentioned above, Gated Recurrent Unit (GRU) simplified LSTM structure and has become an increasingly popular RNN variation.

GRU Compared with LSTM, GRU merges cell state into hidden one, and only uses one gate to control the update of hidden state. Moreover, regarding computation of $\tilde{h}_{t-1}$, GRU sets gate during vector concatenation rather then after tanh activation. (See Figure 12)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 12: The Repeating Module of GRU

All three modules can be the building block of multi-layer RNN and thus meet more complicated real world problems' need. When stacking the model, it only requires to replace the x_t of next RNN with h_t of previous one. (See Figure 13)

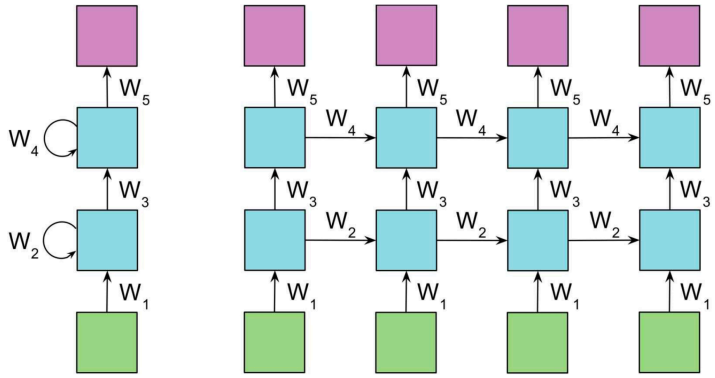


Figure 13: Multi-Layer RNN

3 XGBoost

For a given data set with n examples and m features $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$ ($|\mathcal{D}| = n, \boldsymbol{x}_i \in \boldsymbol{R}^m, y_i \in R$)[5], XGBoost uses K additive functions to make predictions.

$$\hat{y}_i = \phi(\boldsymbol{x}_i) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i), f_k \in \mathcal{F}, \tag{1}$$

where $\mathcal{F} = \{f(\boldsymbol{x}) = w_{q(\boldsymbol{x})}\}$ ($q : \boldsymbol{R}^m \to \{1, 2, \ldots, T\}, \boldsymbol{w} \in \boldsymbol{R}^T$) denotes the function space of trees, q maps the input to a specific leaf node, T indicates the total number of leaves, and $w_q(x)$ is the node weight on leaf $q(\boldsymbol{x})$. With the Eq.(1), we can construct the following objective function $\mathcal{L}$ for optimization with a penalty Ω.

$$\begin{aligned} \mathcal{L}(\phi) &= \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \\ \Omega(f) &= \gamma T + \frac{1}{2}\lambda ||w||_2^2, \end{aligned} \tag{2}$$

Where $l(\cdot, \cdot)$ denotes the loss function for a specified estimating mission, such as square loss for regression or cross entropy for classification.

Since the optimization objective $\mathcal{L}$ contains functions as parameters, we hereby achieve the best ϕ via additive and greedy manner as follow:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t), t = 1, 2, \ldots, K \tag{3}$$

[5] You may refer to the original paper written by Tianqi Chen for thorough introduction.

The Eq.(3) can be approximated by its second order Taylor expansion for the computational convenience.

$$\mathcal{L}^{(t)} \backsimeq \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)] + \Omega(f_t), \tag{4}$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$. Define $I_j = \{i|q(\boldsymbol{x}_i) = j\}, j = 1, 2, \ldots, T$ and remove constant terms in the Eq.(4), we obtain the following t step objective $\tilde{\mathcal{L}}^{(t)}$:

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)] + \Omega(f_t) \\
&= \sum_{j=1}^{T}\sum_{i\in I_j}[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)] + \gamma T + \frac{1}{2}||w||_2^2 \\
&= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T.
\end{aligned} \tag{5}$$

For a given tree structure (given T), the optimal node weight will be

$$w_j^\star = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}, j = 1, 2, \ldots, T. \tag{6}$$

So correspondingly, Eq.(5) will be rewritten as follow,

$$\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{7}$$

The aforementioned equation can be used to evaluate the quality of a tree and thus be a splitting criterion for tree growing at time step t. To be more specific, assume I_L and I_R be the left and right sample set after splitting and let $I = I_L \cup I_R$, then the objective reduction in such process will be

$$\Delta\mathcal{L}_{split} = \frac{1}{2}[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}] - \gamma. \tag{8}$$

Once the tree f_t is grown, it will multiply a shrinkage coefficient (similar with learning rate in stochastic optimization) η to reduce the influence of the tree and its leaves to ϕ, that is,

$$\phi^{(t)} = \phi^{(t-1)} + \eta f_t. \tag{9}$$

Usually a small value η is preferred associated with a shallow tree structure.

The remaining question now is the split finding algorithm for each node in a tree every time step. For the sake of getting intuition, I only post the traditional exact greedy version algorithm here though XGBoost make quite a lot refinements in this part.

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0, H_L \leftarrow 0$
  for j *in sorted(I,by* $\boldsymbol{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
    score$\leftarrow \max(\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

In practice, there are a lot of hyperparameters that need to be tuned, this introduction may help you better understand the concrete meanings behind some hyperparameters. As for practical suggestions of tuning, this tutorial may deserve your attention and time.

4 End Notes

In this short report, I review the main ideas of neural network, RNN with its variants and XGBoost. I hope it can bring you some fast intuition about aforementioned methods and be a good reference when you try to apply some of them. It will be highly appreciated if you would like to make some contributions to refine this report. Please contact me via bobzht341@gmail.com if you have any questions or suggestions.